Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation
CRT-2006-15

A TWO-LEVEL SIMULATED ANNEALING ALGORITHM FOR EFFICIENT DISSEMINATION OF ELECTRONIC CONTENT

by

Tolga Bektas
Jean-François Cordeau
Erhan Erkut
Gilbert Laporte

July 2006

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone: (514) 343-7575 / Télécopieur: (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

106-06870/

PUBLICATION
CRT-2006-15

A TWO-LEVEL SIMULATED ANNEALING ALGORITHM FOR EFFICIENT DISSEMINATION OF ELECTRONIC CONTENT

by

Tolga Bektas[1,2*]
Jean-François Cordeau[1]
Erhan Erkut[2]
Gilbert Laporte[1]

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 and Canada Research Chair in Distribution Management, HEC Montréal, 3000, Côte-Sainte-Catherine, Montréal, Canada H3T 2A7 (tolga,cordeau,gilbert@crt.umontreal.ca)

2. University of Alberta, School of Business, Edmonton, Alberta, Canada T6G 2R2 (Erhan.Erkut@UAlberta.Ca)

* Corresponding author: tolga@crt.umontreal.ca

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2006

Abstract

This paper describes a two-level simulated annealing heuristic for the joint problem of object placement and request routing in a Content Distribution Network (CDN) with constraints pertaining to server capacity and end-to-end object transfer delay. A CDN is a technology that is used to efficiently distribute electronic content throughout an existing network. The two-level solution procedure is composed of an exterior and an interior simulated annealing algorithms. Computational experiments are conducted by comparing the proposed procedure with several truncated exact solution algorithms on randomly generated Internet topologies.

Keywords: telecommunications, content distribution network, object placement, request routing, simulated annealing.

Résumé

Cet article décrit une heuristique pour le problème conjoint d'affectation de contenu et de routage des requêtes dans un réseau de distribution de contenu électronique. Des contraintes sont imposées sur la capacité des serveurs et sur les délais de transmission de l'information. L'heuristique proposée est un algorithme de recuit simulé à deux niveaux qui traite simultanément l'affectation du contenu aux serveurs et le routage des requêtes. Cet algorithme est comparé à des méthodes de résolution exactes sur des données générées aléatoirement.

Mots-clés: télécommunications, réseau de distribution de contenu, affectation, routage, recuit simulé.

Introduction

The increasing popularity of the World Wide Web and the high demand for electronic information are now posing new challenges to the efficient distribution of electronic content. The growth of the size of the delivered content and the number of users have resulted in unacceptable delays for end users, increased loads for servers and network congestion. The delays experienced by end users have important economic consequences, especially in electronic marketing. A widely observed behaviour is that a typical client will abandon a Web site failing to download in less than eight seconds. According to a report by Zona Research conducted in 1999, "the amount of time taken for Web pages to load is one of the most critical factors in determining the success of a site and the satisfaction of its users" (`http://also.co.uk/e-hosting.html` accessed 26.06.2006). In the same report, the potential losses due to unacceptably slow response times was estimated to be about \$4.35 billion in 1999. In a similar research conducted two years later, Zona Research (`http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf` accessed 26.06.2006) reported the corresponding figure for 2001 to be over \$25 billion.

For these reasons, distribution of electronic content has become an important problem. Efforts to overcome delays and to alleviate the internet traffic have given rise to the new technology called *Content Distribution* (or *Delivery*) *Networks* (CDNs). The main objective of CDNs is to replicate the content from the origin server to geographically distributed *proxy servers*, which act on behalf of the origin server by delivering the requested content to the clients. CDNs therefore aim at moving the content as close as possible to the clients. A CDN can significantly improve the performance of a network and reduce the total cost associated with distributing content, since the clients are no longer served by the origin server, but from a nearby proxy server instead.

Ideally, one would like to fully replicate the whole content to every proxy server in the CDN. This is, in most cases, not possible due to the limited capacity of the proxy servers as well as the high cost of replication. Therefore, a CDN must adequately replicate the subsets of the content in an intelligent manner, so as to efficiently use the network resources. This problem is referred to as the *object placement* (or *object replication*) problem. Another challenging task for a CDN is, given a request from a client, to identify the best proxy server that should respond to this request. This is referred to as *request routing*. Both of these problems are interrelated, and should therefore be considered jointly for a CDN to operate in an efficient manner. Additionally, commercial CDNs also have the problem of guaranteeing a limit on end-to-end object transfer delay. This is generally part of the Quality of Service (QoS) requirements and is specified in the Service Level Agreement (SLA), usually negotiated and mutually agreed upon by the CDN itself and its customers.

There already is a significant amount of research on CDNs, most of them being due to the computer science community. In what follows, we will only mention some that are of interest to the present study. One of the first topics studied in CDN research is the proxy server location problem, which consists of optimally locating a number of proxy servers in a network and assigning each client to an established proxy server so as to minimize the total cost of location and assignment. We refer the reader to Li et al. (1999) and Qiu et al. (2001) for details. For the object placement problem, Cidon et al. (2002), Kangasharju et al. (2002) and Yang and Fei (2003) are good references. The object placement problem is in some ways similar to the *database location problem* in computer communications networks. The reader may refer to Fisher and Hochbaum (1980), Pirkul (1986), Gavish and Suh (1992), Chari (1996) and Hakimi and Schmeichel (1997) for additional studies on the database location problem.

There exist a number of papers in which several problems within a CDN are addressed simultaneously. For example, Ryoo and Panwar (2001) study the problem of distributing multimedia files in networks involving the determination of the communication link capacities, size of the multimedia servers, and distribution of different types of content to each server. In another study, Xu et al. (2002) investigate the problem of determining the optimal number and location of proxies along with the placement of replicas of a single object on the installed proxies, given a maximum number of potential proxies and a tree-like topology. Xuanping et al. (2003) discuss the joint problem of proxy server placement and object placement in a CDN, subject to a budget constraint. The authors assume that each client is assigned to its closest proxy. In a similar context, Laoutaris et al. (2004) consider the joint problem of optimal location of the objects together with the capacity dimensioning of the proxies. Another study by Laoutaris et al. (2005) addresses the storage capacity allocation problem for CDNs, which takes into account the optimal location and capacity of the proxies, and the objects that should be placed in each of them. The assignment of clients is not considered as a decision problem, since the authors assume a given hierarchial topology where the assignment of clients is predetermined. In a relatively recent study, Almeida et al. (2004) have considered the problem of jointly routing requests and placing proxy servers in streaming CDNs. More recently, Nguyen et al. (2005) have analyzed the problem of provisioning so called Overlay Distribution Networks, which includes proxy server placement, request routing, and object replication.

Bektaş et al. (2007) have addressed a design problem arising in CDNs and involving three related aspects: (i) the number and the location of the proxy servers to be used in the CDN among a given set of potential sites (*proxy server placement*), (ii) the objects to be located in each proxy server (*object placement*), and (iii) the assignment of each client to a proxy server (*request routing*) while taking into account capacity constraints on the proxy servers. Recently, Bektaş et al. (2006) have proposed a nonlinear integer programming formulation for the joint problem of object placement and request routing in CDNs incorporating capacity and QoS constraints on object transfer time. The authors have described three linearization procedures. Based on two of the linearized formulations, they have also proposed two exact solution algorithms based on Benders decomposition and Lagrangean relaxation. Although the algorithms proposed by Bektaş et al. (2006) were able to produce better solutions than those obtained by CPLEX with a given time limit, the problem proves to be hard to solve to optimality due to the size of the instances that may arise in practice. Powerful heuristics are therefore necessary for the solution of realistic instances.

The aim of this paper is to describe a heuristic algorithm for the problem described above. The proposed algorithm is a two-level implementation of the well known simulated annealing metaheuristic (Kirkpatrick et al., 1983; Cerny, 1985), and is simple, easy to implement, and able to produce good quality solutions within reasonable computing times.

The paper is organized as follows. We first formally define the problem in the next section. To provide a clear description of the problem, this section also includes the integer programming formulation proposed by Bektaş et al. (2006). A two-level simulated annealing algorithm for the problem is then described. Results of computational experiments performed on randomly generated test problems are reported, followed by conclusions.

Problem Definition

To formulate the problem, we assume a given complete network $G = (V, E)$, where V is the set of nodes and $E = \{\{i, j\} : i, j \in V\}$ is the set of edges. The node set V is partitioned into $\{I, J, S = \{0\}\}$, where I is the set of clients, J is the set of nodes on which proxy servers are installed, and S is a singleton containing the origin server. We also assume without loss of generality that no client can directly access the origin server (e.g., for security reasons). A typical CDN architecture with a single origin server, three proxy servers and nine clients is depicted in Figure 1. Given a client-proxy pair (i, j) with $i \in I$ and $j \in J$, we denote by c_{ij} the cost of transferring an object over a client-proxy pair (i, j). We denote by d_{ij} the delay representing the amount of time required to retrieve data between nodes i and j. While appropriate queueing models may be used to measure the delay in the network, we assume here that the average delay for each link is known, as in Nguyen et al. (2005) and Wauters et al. (2006), and can be calculated as the sum of propagation and transmission delays (Erçetin and Tassiulas, 2003). Since we assume that an existing network is in place, we ignore the option of network expansion. We further assume that the capacity of the potential server at site j is s_j. We define K as the set of objects located in the origin server and assume that the size of each object $k \in K$ is b_k. Also we consider that the probability of client $i \in I$ requesting object $k \in K$ over a given time interval is denoted by λ_{ik}.

Insert Figure 1 about here.

The time required to transfer object k from proxy server j to node i is considered to be a function of object size (b_k) and the "distance" between nodes i and j, and denoted by $t(b_k, d_{ij})$. Here, distance can be considered as the number of hops or unit latency between the two nodes. Similarly, the time needed to transfer an object k from the origin server to a node i via proxy j is denoted by $t'(b_k, d_{ij})$. We assume without loss of generality that $t(b_k, d_{ij}) < t'(b_k, d_{ij})$. Finally, we denote by Δ_{ik} the upper bound on the end-to-end object transfer delay for client i to access object k. A summary of the notation used in defining the problem is given in Table 1.

Insert Table 1 about here.

The problem considered in this study is to minimize the total cost of distributing the content under the following three constraint classes:

1. each client is to be assigned to a single proxy server;
2. a set of objects is to be placed in each proxy server without exceeding the available capacity;
3. the total delay for a client to access an object can not exceed the QoS requirement.

To formulate the problem under consideration, we define the following binary decision variables:

$$x_{ij} = \begin{cases} 1, & \text{if client } i \in I \text{ is assigned to proxy server } j \in J \\ 0, & \text{otherwise} \end{cases}$$

$$z_{jk} = \begin{cases} 1, & \text{if proxy server } j \in J \text{ holds object } k \in K \\ 0, & \text{otherwise.} \end{cases}$$

The formulation is as follows:

$$(\mathcal{P}) \quad \text{Minimize} \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\Big(b_k\lambda_{ik}c_{ij}x_{ij}z_{jk} + b_k\lambda_{ik}(c_{ij}+c_{j0})x_{ij}(1-z_{jk})\Big) \tag{1}$$

subject to

$$\sum_{j\in J} x_{ij} = 1 \qquad \forall i \in I \tag{2}$$

$$\sum_{k\in K} b_k z_{jk} \leq s_j \qquad \forall j \in J \tag{3}$$

$$\sum_{j\in J} t(b_k, d_{ij})x_{ij}z_{jk} + \sum_{j\in J} t'(b_k, d_{ij})x_{ij}(1-z_{jk}) \leq \Delta_{ik} \qquad \forall i \in I, k \in K \tag{4}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, j \in J \tag{5}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K. \tag{6}$$

The first part of the objective function (1) denotes the total cost of transferring the content for the case where client i receives content k that is located in proxy j (reflected by the cost $b_k\lambda_{ik}c_{ij}$ summed over all the proxies, clients and objects). In the case where the requested object is not located in a proxy server, an additional cost is incurred to further request the object from the origin server. This is reflected in the second part of the summation by c_{j0}. Constraints (2) imply that a client can only be assigned to a single proxy server. Constraints (3) imply that the total size of objects held in each proxy server is constrained by the available capacity. Using constraints (4), we model the condition that the total time required to transfer object k to a client node j from either a proxy server j ($t(b_k, d_{ij})$) or from the origin server ($t'(b_k, d_{ij})$) can not exceed the upper bound Δ_{ik}.

A recent study by Bektaş et al. (2006) proves that the problem is $\mathcal{NP}$-hard and shows that neither a branch-and-cut algorithm nor two decomposition based exact algorithms succeed in solving $\mathcal{P}$ optimally for large instances within reasonable time limits. Bearing in mind the dynamic context of the environment in which this problem arises, there is a need to develop a fast algorithm for the solution of the problem that is capable of producing good quality solutions. Based on this motivation, we propose a simulated annealing algorithm for the solution of this problem and describe it in the next section.

A Two-Level Simulated Annealing Algorithm

The simulated annealing algorithm proposed here consists of two levels. More specifically, we use an "exterior" simulated annealing algorithm at the first level. This algorithm randomly generates a neighbour at every iteration. An "interior" simulated annealing algorithm at the second level is then used to improve this solution. Before going into the details, some necessary definitions for the description of the algorithm will be stated below.

Define, for each pair (i,j) the sets $\mathcal{R}_{ij} = \{k \in K | t(b_k, d_{ij}) > \Delta_d\}$, $\mathcal{Q}_{ij} = \{k \in K | t(b_k, d_{ij}) \leq \Delta_d$ and $t'(b_k, d_{ij}) > \Delta_d\}$, and $\mathcal{G} = \{(i,j) | i \in I, j \in J$ and $\mathcal{R}_{ij} \neq \emptyset\}$. Observe that for every pair

$(i,j) \in \mathcal{G}$, one can set $x_{ij} = 0$ since this implies that there exists at least one $k^* \in \mathcal{R}_{ij}$ for which client i cannot be served by proxy j without exceeding the delay limit. We further define for each $i \in I$ the sets $\mathcal{F}_i = \{j \in J | \mathcal{Q}_{ij} = \mathcal{R}_{ij} = \emptyset\}$ and $\mathcal{H}_i = \{j \in J | z^*_{jk} = 1, \forall k \in \mathcal{Q}_{ij}\}$. We now state a proposition that will be used in the development of the algorithm.

Proposition 1 *Let $\mathcal{Z} = \{z \in \mathbf{B}^{|J||K|} | z$ satisfies (3)$\}$. For any given $z \in \mathcal{Z}$, the associated request routing subproblem can be solved by inspection.*

Proof The proposition is based on the fact that, for a given set of $z^* \in \mathcal{Z}$, $\mathcal{P}$ becomes

$$(\tilde{\mathcal{P}}) \quad \text{Minimize} \quad \sum_{i \in I} \sum_{j \in \mathcal{F}_i \bigcup \mathcal{H}_i} \hat{c}_{ij} x_{ij}$$

subject to

$$\sum_{j \in \mathcal{F}_i \bigcup \mathcal{H}_i} x_{ij} = 1 \qquad \forall i \in I$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, j \in \mathcal{F}_i \cup \mathcal{H}_i,$$

where $\hat{c}_{ij} = \sum_{k \in K} (b_k \lambda_{ik} (c_{ij} + c_{j0}(1 - z^*_{jk}))$. It is easy to see that $\tilde{\mathcal{P}}$ can be solved by assigning each $i \in I$ to any $p \in \underset{j \in \mathcal{F}_i \bigcup \mathcal{H}_i}{\text{argmin}} \{c_{ij}\}$. The latter follows from the fact that a client $i \in I$ can only be assigned to $j \in \mathcal{F}_i \bigcup \mathcal{H}_i$ so as to respect the QoS requirement.□

The proof of Proposition 1 provides, for a given object placement scheme, a simple $\mathcal{O}(|I||J|)$ time procedure to assign the clients to the proxy servers. We denote this procedure by CLIENTASSIGN.

The development of the two-level simulated annealing algorithm is based on two types of neighbourhoods, which will be described below. A general outline of the two-level simulated annealing algorithm is given in Algorithm 1. In this algorithm, $v(\mathcal{S})$ and $\mathcal{N}(\mathcal{S})$ respectively denote the solution value and the set of neighbours of a solution $\mathcal{S}$. We will also denote by $L(j)$ the index set of objects already located in proxy server j, $\bar{L}(j) = K \setminus L(j)$ and $b(L(j)) = \sum_{k \in L(j)} b_k$. T_1 and T_2 are used to denote the temperature for the first and second levels, respectively. Similarly, T_1^f and T_2^f are the final temperatures at which the first and the second level algorithms are stopped, respectively. Finally, α is the cooling rate at which the temperature is decreased at every iteration.

Generation of the first level neighbourhood

Given any solution $\mathcal{S}$, the first type of neighbourhood we consider is obtained by changing the object placement scheme of S while respecting the capacity constraints (3), which is followed by the CLIENTASSIGN procedure, where each client is assigned to a proxy server while satisfying the assignment (2) and delay (4) constraints. Note that a neighbour $\mathcal{S}' \in \mathcal{N}(\mathcal{S})$ may have a different client routing structure than that of $\mathcal{S}$.

Note that when the numbers of proxy servers ($|J|$) and objects ($|K|$) are high, a single execution of this procedure may require significant computational effort. To reduce the effort, we consider only a random subset of proxy servers (denoted by $\hat{J}$) and objects (denoted by $\hat{K}$) at each iteration.

A general outline of generating a level one neighbour is given in Algorithm 2.

Algorithm 1 Two-Level Simulated Annealing Algorithm

Generate an initial starting solution $\mathcal{S}_1$ and set the best known solution $\hat{\mathcal{S}}_1 = \mathcal{S}_1$. Initialize the starting temperature $T_1 = T_1^i$.
while $T_1 \geq T_1^f$ **do**
  Choose a random solution $\mathcal{S}_1^r \in \mathcal{N}(\mathcal{S}_1)$ and let $\Delta = v(\mathcal{S}_1^r) - v(\mathcal{S}_1)$.
  if $\Delta \leq 0$ **then**
    Let $\mathcal{S}_1^r = \mathcal{S}_1$. If $v(\mathcal{S}_1^r) \leq v(\hat{\mathcal{S}}_1)$, then $\hat{\mathcal{S}}_1 = \mathcal{S}_1^r$.
  else
    Let $u \sim U(0,1)$. If $u < e^{-\Delta/T_1}$, then $\hat{\mathcal{S}}_1 = \mathcal{S}_1^r$.
  end if
  Let $\mathcal{S}_2 = \hat{\mathcal{S}}^1$ and set the best known solution $\hat{\mathcal{S}}_2 = \hat{\mathcal{S}}_1$. Initialize the starting temperature $T_2 = T_2^i$.
  while $T_2 \geq T_2^f$ **do**
    Choose a random solution $\mathcal{S}_2^r \in \mathcal{N}(\mathcal{S}_2)$ and let $\Delta = v(\mathcal{S}_2^r) - v(\mathcal{S}_2)$.
    if $\Delta \leq 0$ **then**
      Let $\mathcal{S}_2^r = \mathcal{S}_2$. If $v(\mathcal{S}_2^r) \leq v(\hat{\mathcal{S}}_2)$, then $\hat{\mathcal{S}}_2 = \mathcal{S}_2^r$.
    else
      Let $u \sim U(0,1)$. If $u < e^{-\Delta/T_2}$, then $\hat{\mathcal{S}}_2 = \mathcal{S}_2^r$.
    end if
    $T_2 = \alpha T_2$
  end while
  $T_1 = \alpha T_1$
end while

Algorithm 2 Generation of a Level I neighbour

for all $j \in \hat{J}$ **do**
  sort objects $k \in L(j)$ as $O(j) = \{o_1, o_2, \ldots\}$.
  sort objects $k \in \bar{L}(j)$ as $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$.
  $t := 1$
  while $t \leq |\hat{K}|$ **do**
    $L(j) := L(j) \setminus \{o_t\}$
    $r := 1$
    while $r \leq |\bar{O}(j)|$ **do**
      if $b(L(j) \cup \{\bar{o}_r\}) \leq s_j$ **then**
        $L(j) := L(j) \cup \{\bar{o}_r\}$
      end if
      $r \leftarrow r + 1$
    end while
    $t \leftarrow t + 1$
  end while
end for
Perform CLIENTASSIGN

Generation of the second level neighbourhood

Each neighbour generated in the first level of the main algorithm can be improved by modifying the object placement scheme while preserving the assignment structure. Such a neighbour can be obtained by a removal (addition) of an object to (from) a proxy server by respecting the QoS constraints. This may be accomplished by means of a local search heuristic. We thus propose to use an "interior" simulated annealing algorithm at each iteration of the "exterior" algorithm in the hope of obtaining better solutions.

Note that in defining such a neighbour, one can only add or remove objects which would not result in the delay constraints (4) being violated. Such objects are those in the set Q_{ij} for each pair (i, j) with $i \in I$ and $j \in \mathcal{H}_i$ for which $x_{ij} = 1$.

In contrast to the Level I neighbourhood, Level II neighbourhoods always have the same client routing structure for all the neighbours $\mathcal{N}(\mathcal{S})$ of a solution $\mathcal{S}$, but the object placement may differ. A general outline of generating such a neighbour is given in Algorithm 3.

Algorithm 3 Generation of a Level II neighbour

for all $j \in \hat{J}$ **do**
  sort objects $k \in L(j) \cap Q_{ij}$ as $O(j) = \{o_1, o_2, \ldots\}$.
  sort objects $k \in \bar{L}(j)$ as $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$.
  $t := 1$
  while $t \leq |\hat{K}|$ **do**
    $L(j) := L(j) \setminus \{o_t\}$
    $r := 1$
    while $r \leq |\bar{O}(j)|$ **do**
      if $b(L(j) \cup \{\bar{o}_r\}) \leq s_j$ **then**
        $L(j) := L(j) \cup \{\bar{o}_r\}$
      end if
      $r \leftarrow r + 1$
    end while
    $t \leftarrow t + 1$
  end while
end for

Orderings for object removal and insertion

As we have previously stated, the neighbourhood generation process during the simulated annealing algorithm is based on a modification of the object placement scheme. Such a modification may be accomplished by removing a set of objects from a given proxy server, which can then be followed by adding a new set of objects to the same server. The manner in which the objects are removed and inserted may be based on several criteria. We describe those we have chosen to use in this study. We start by introducing three criteria for the removal of objects from a given server.

1. *Removal by size*: In this case the objects are removed in decreasing order of their size. The main idea behind this criterion is to remove the objects that occupy much of the space in a

proxy server to be able to replace them with more "profitable" objects later on. Using size as a criterion, the sorting order would be $O(j) = \{o_1, o_2, \ldots\}$, where $b_{o_k} \geq b_{o_{k+1}}$ for $k \in L(j)$.

2. *Removal by popularity*: Another criterion we consider for object removal is based on *global object popularity*, calculated as:

$$\psi_k = \sum_{i \in I} d_{ik}. \tag{7}$$

In this case, objects are removed in the increasing order of popularity, and hence the order may be given as $O(j) = \{o_1, o_2, \ldots\}$, where $\psi_{o_k} \leq \psi_{o_{k+1}}$ for $k \in L(j)$.

3. *Removal by savings*: The final criterion we consider for object removal is based on *object savings*. Defining $I_j = \{i \in I | x_{ij} = 1\}$ for every $j \in J$, the savings of an object k for a given proxy server j can be calculated as

$$\pi_{jk} = \sum_{i \in I_j} b_k \lambda_{ik} c_{j0}. \tag{8}$$

A saving π_{jk} for a server-object pair (j, k) denotes the amount of cost reduction achieved by placing object k in server j. In this case, the order is determined by $O(j) = \{o_1, o_2, \ldots\}$ where $\pi_{j,o_k} \leq \pi_{j,o_{k+1}}$ for $k \in L(j)$.

Having defined several criteria for object removal, we now present four other criteria for object insertion.

1. *Insertion by popularity*: The first criterion is based on the global object popularity given in (7). In this case, the insertion ordering may be given as $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$, where $\psi_{o_k} \geq \psi_{o_{k+1}}$ $k \in \bar{L}(j)$.

2. *Insertion by unit popularity*: The second criterion used for inserting objects is the popularity to size ratio defined by

$$\rho_k = \psi_k / b_k. \tag{9}$$

In this case, the insertion ordering would be $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$, where $\rho_{o_k} \geq \rho_{o_{k+1}}$ for $k \in \bar{L}(j)$.

3. *Insertion by savings*: The third criterion used for inserting objects is the saving defined by (8). In this case, the insertion ordering is given by $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$ where $\pi_{j,o_k} \geq \pi_{j,o_{k+1}}$ for $k \in \bar{L}(j)$.

4. *Insertion by local popularity*: We now introduce the concept of *local popularity* for an object. More specifically, the local popularity of an object k with regards to a proxy server j measures how much the object is requested by the clients assigned to this proxy server and calculated as follows:

$$\phi_{jk} = \sum_{i \in I_j} \lambda_{ik}. \tag{10}$$

In this case, the insertion ordering is given as $\bar{O}(j) = \{\bar{o}_1, \bar{o}_2, \ldots\}$, where $\phi_{j,o_k} \geq \phi_{j,o_{k+1}}$ for $k \in \bar{L}(j)$.

Generation of the initial solution

We propose a simple procedure to generate an initial solution. The procedure consists of ordering all the objects with respect to an insertion criterion, and placing them on each proxy in this order until the capacities are full. Although any of the four criteria presented above may be used for this purpose, our preliminary testing suggested that the popularity criterion performs better than the other three. It is therefore used in generating the initial solution. Once the object placement is performed in this manner, client assignments are made through the procedure CLIENTASSIGN. The outline of this procedure is given in Algorithm 4.

Algorithm 4 Generation of Initial Solutions

sort objects $k \in K$ as $G = \{g_1, g_2, ..., g_{|K|}\}$ where $\psi_{g_k} \geq \psi_{g_{k+1}}$ for $k = 1, \ldots, |K| - 1$
for all $j \in J$ **do**
  $L(j) := \emptyset$, $t := 1$
  while $t \leq K$ **do**
    if $b(L(j) \cup \{g_t\}) \leq s_j$ **then**
      $L(j) := L(j) \cup \{g_t\}$
    end if
    $t \leftarrow t + 1$
  end while
end for
Perform CLIENTASSIGN

Computational Experiments

The algorithm just described was coded in C and run on a a 3GHz Pentium 4 workstation running Linux. This section includes the details of the computational experiments. The first section describes how test instances were generated whereas the second section presents the results obtained with the proposed algorithm.

Generation of test instances

Computational experiments were performed using randomly generated data, which were described by Bektaş et al. (2006). Here, we describe the generation process in full for the sake of completeness.

Although it is not easy to obtain practical data for a problem that arises in a very dynamic and a relatively unstable environment (such as the Internet), it is to some extent possible to incorporate some real-life aspects of a content distribution environment in the data generation. First, the topologies used for the experiments were generated using an Internet topology generator GT-ITM (available at `http://www.cc.gatech.edu/projects/gtitm`) that mimics the characteristics of the real Internet topology. The unit transfer cost between nodes i and j was interpreted as the number of hops (edges) between these nodes. While any other parameter, such as the unit bandwidth cost, can be used as the cost metric, these may be hard to obtain in practice. In contrast, the hop count between two nodes is relatively easy to obtain and is stated to be "a decent indicator of the path's proximity, reliability, and stability" (Qiu et al., 2001). Moreover, it is generally assumed in the literature that the amount of latency (or delay) between two nodes is proportional to the number

of hops between these two nodes (Radoslavov et al., 2002). Further, Huang and Abdelzaher (2005) report that the "average network latency of downloading a file is roughly proportional to its size when the file size is between 1KB and 100KB", where the reported range contains a significant portion of Web objects (Saroiu et al., 2002). We have therefore generated the size of each object to be a continuous uniform random variable between 1 and 100. The capacity of each proxy server was defined as 40% of the total size of all objects. To determine the latency parameters, we have defined object groups in terms of their sizes and associated a nonnegative value for both unit delay and the latency bound, which can be seen in Table 2. For simplicity, we assume that the unit delay u_l^k is the same for a given content class and given in milliseconds (ms). The delay caused by transferring an object k over link (i, j) is then calculated as $t(b_k, d_{ij}) = u_l^k d_{ij}$, where $d_{ij} = \sigma c_{ij}$ and σ is a constant. Similarly, the latency bound Δ_d is the same for all objects in a specific content group and is also given in milliseconds. We assume that the minimum delay that a client experiences to reach the nearest server in retrieving an object is 20ms (see the measurement results given by Johnson et al. (2001) on two major commercial CDNs) and increases with object size.

Insert Table 2 about here.

It it well known (see Breslau et al., 1999) that the demand distribution of Web requests follows a Zipf-like distribution (Zipf, 1949). We have therefore used this distribution when generating the object request patterns. Let a set of objects be ranked in order of their popularity where object i in this order is the i^{th} most popular object. The Zipf-like distribution assumes that the probability of a request for an object is inversely proportional to $i^{-\alpha}$. More specifically, given an arrival for a request, the conditional probability that the request is for object i is given as:

$$P_K(i) = \Omega i^{-\alpha},$$

where

$$\Omega = \left(\sum_{j=1}^{K} j^{-\alpha} \right)^{-1}$$

is a normalization constant. When $\alpha = 1$, we have the true Zipf-distribution. In Breslau et al. (1999), it is shown that α varies from 0 to 1 for different access patterns and is usually between 0.64 and 0.83 for Web objects. This value was recorded to be $\alpha = 0.733$ for multimedia files in Yang and Fei (2003). We have used this specific value in our implementation.

Results of the experiments

The first part of the experiments deals with fine tuning of the proposed algorithm by comparing the different removal and insertion criteria described above for generating neighbours. We will hence refer to a given removal and insertion criteria as a modification *policy*. We will denote by (a, b) a policy that uses criterion a to remove and criterion b to insert objects. To refer to each removal and insertion criterion, we will use the symbols that were introduced to define the corresponding criterion. For instance, the designation (ψ, ϕ) refers to the policy where objects are removed in the increasing order of their savings and inserted in the decreasing order of their local popularity.

Given three possible removal and four insertion criteria, there are $3 \times 4 = 12$ possible policies for the first level and the same number for the second. This number is $12 \times 12 = 144$ for all possible combinations for all the policies that may be tested considering the two levels of the algorithm.

Since it is not practical to investigate all these combinations, we test the algorithm in two stages by first determining the best policy for the first level without considering the second level. Given the best performing policy for this level, we then proceed to do a similar testing for the second level.

All computational experiments are performed by using 15 large scale instances in which the number of proxy servers ranges from 20 to 60, the number of objects ranges from 60 to 100, and the number of clients is 200. The sizes of the instances are reported in Table 5.

Fine tuning of level I

We have conducted computational tests to find which policies yield good results for the first level of the algorithm. The parameters of the algorithm for the first level are set as follows: at every iteration, we choose $\hat{J}$ to be a random subset of J with $|\hat{J}| \approx 0.25|J|$ and similarly $\hat{K}$ to be a random subset of K with $|\hat{K}| \approx 0.3|K|$. The cooling rate for this level is set to be $\alpha = 0.95$.

The full results are presented in Tables 7–9 in the Appendix. In each table, we present, for each policy (a, b), the value of the solution obtained under columns $v(a, b)$ by using each policy and the required solution time under columns $t(a, b)$ (in CPU seconds). For convenience, a summary of these results is given in Table 3, where a + sign under a given policy indicates that the best solution value was found using this policy.

Insert Table 3 about here.

The results presented in Table 3 have interesting implications. In general, the best results were obtained with the popularity criterion (ψ) for object removal. Together with this criterion, insertion by savings (π) yielded the the best results for 11 of the instances out of the 15 (about 73%). This is followed by insertion by the unit popularity (for three instances) and insertion by popularity criterion (for one instance). The results also suggest that, in terms of object removal, neither the size (b) nor the savings (π) criteria perform as good as removal by popularity.

Fine tuning of level II

This section presents the results of the computational experimentation performed to investigate the effects of 12 different policies used at the second level of this algorithm. For this purpose, we have fixed the policy at the first level to (ψ, π) based on the results of the previous section. The parameters of the algorithm for the second level are set as follows: At every iteration, we choose $\hat{J}$ to be a random subset of J with $|\hat{J}| \approx 0.5|J|$ and similarly $\hat{K}$ to be a random subset of K with $|\hat{K}| \approx 0.5|K|$. The cooling rate for this level is set to be $\alpha = 0.85$. The full results are presented in Tables 10–12 in the Appendix. A summary of these results is given in Table 4.

Insert Table 4 about here.

Tables 3 and 4 suggest that the savings criterion is significantly better than the others for the object removal (10 out of 15 instances). In general, the policy (π, ρ) performs better on seven of the 15 instances (about 47%).

Comparison with other algorithms

In this section, we test the efficiency of the proposed algorithm by comparing the results with those obtained by a one-level SA algorithm and several truncated exact methods. The comparison with a standard "one-level" implementation of the simulated annealing algorithm is performed in order to see the effect of the addition of the second level. In this one-level SA implementation, we use both level one and two neighbourhoods described above. We assign probabilities to each of these neighbourhoods and pick one of them accordingly at each iteration. The parameters used in this implementation are those found as a result of the fine-tuning process described above. To find the best setting of the weight parameters, we have performed additional experiments. More specifically, the probabilities of the neighbourhoods are varied from 0.1 to 0.9, in increments of 0.1. Full results of this experiment are given in Table 5, where the first three columns are as previously stated. The next nine columns (in the format p_1/p_2) denote the solution value obtained by assigning probabilities p_1 and p_2 for the level one and level two neighbourhoods, respectively. The results of these experiments suggest that the level one neighbourhood plays an important role in obtaining good solutions.

Insert Table 5 about here.

The exact solution methods used in the comparisons are those proposed by Bektaş et al. (2006) and consist of a Benders decomposition algorithm, a Lagrangian relaxation and decomposition algorithm, and a simplex based branch-and-cut algorithm.

The results comparing the proposed algorithm with those stated above are presented in Table 6. Column v_1 lists the best results obtained by one-level SA algorithm over all possible weight settings. Column v_2 corresponds to the best solution values obtained by the two-level simulated annealing algorithm, over all possible combinations of different criteria. The next column titled v_e gives the best upper bounds obtained as a result of running the exact solution procedures for one hour on the same computer. The last column indicates, for each instance, by how much the solution found by the proposed simulated annealing algorithm improves over the "truncated exact solution value". This improvement is calculated as $100(v_e - v_2)/v_e$.

Insert Table 6 about here.

The results presented in Table 6 have two main implications. First, the results show that the two-level simulated annealing algorithm is capable of improving the results found by a one-level algorithm for all of the instances. The second implication is that the proposed algorithm yields better solutions than the truncated exact solution algorithms for instances with $|J| \geq 30$. The fact that exact solution algorithms produce better solutions for instances with $|J| = 20$ is due to the small number of proxy servers, which reduces the model size and makes the exact methods superior. However, as the number of servers increases (which is generally the case with commercial CDN providers), the proposed algorithm is superior to the truncated exact methods *both* in terms of the solution value and of the required computational effort. More specifically, the improvements in the solution value can be as high as 8%. Moreover, the time required by a single run of the proposed algorithm is usually less than ten minutes, which is one tenth of the time required by *any* of the "truncated" exact solution methods.

Conclusions

We have presented a metaheuristic algorithm for the solution of the joint object placement and request routing problem in a Content Distribution Network, taking into account capacity and certain QoS constraints. The proposed procedure is a two-level simulated annealing algorithm which is reasonably simple and easy to implement. Computational results demonstrate that the proposed algorithm is superior to a fairly standard implementation of a one-level simulated annealing algorithm and to several truncated exact solution methods, both in terms of solution quality and of computational effort.

Acknowledgments

The first author is grateful to Iradj Ouveysi for introducing him to the topic of content distribution, and to Vitória Pureza for helpful discussions on the simulated annealing algorithm. This research has been partially supported by a Strategic Project Grant of the Natural Sciences and Engineering Research Council of Canada (NSERC). We gratefully acknowledge this support.

References

J.M. Almeida, D.L. Eager, M.K. Vernon, and S.J. Wright. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE T Multimedia*, 6:356–365, 2004.

T. Bektaş, J.-F. Cordeau, E. Erkut, and G. Laporte. Exact algorithms for the joint object placement and request routing problem in content distribution networks. 2006. Center for Research on Transportation, Publication CRT-2006-03.

T. Bektaş, O. Oğuz, and I. Ouveysi. Designing cost-effective content distribution networks. *Comput Oper Res*, 2007. Forthcoming.

L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *Proceedings of IEEE INFOCOM'99*, volume 1, pages 126–134, New York, 1999.

V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J Optimiz Theory App*, 45:41–51, 1985.

K. Chari. Resource allocation and capacity assignment in distributed systems. *Comput Oper Res*, 23:1025–1041, 1996.

I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Comput Netw*, 40: 205–218, 2002.

Ö. Erçetin and L. Tassiulas. Market-based resource allocation for content delivery in the Internet. *IEEE T Comput*, 52:1573–1585, 2003.

M.L. Fisher and D.S. Hochbaum. Database location in computer networks. *J Assoc Comput Mach*, 27:718–735, 1980.

B. Gavish and M.W. Suh. Configuration of fully replicated distributed database system over wide area networks. *Ann Oper Res*, 36:167–192, 1992.

S.L. Hakimi and E.F. Schmeichel. Locating replicas of a database on a network. *Networks*, 30: 31–36, 1997.

C. Huang and T. Abdelzaher. Bounded-latency content distribution: Feasibility and evaluation. *IEEE T Comput*, 54:1422–1437, 2005.

K.L. Johnson, M.S. Carr, J.F.and Day, and M.F. Kaashoek. The measured performance of content distribution networks. *Comput Commun*, 24:202–206, 2001.

J. Kangasharju, J. Roberts, and K.W. Ross. Object replication strategies in content distribution networks. *Comput Commun*, 25:376–383, 2002.

S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–680, 1983.

N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. Joint object placement and node dimensioning for Internet content distribution. *Inform Process Lett*, 89:273–279, 2004.

N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Comput Netw*, 47:409–428, 2005.

B. Li, M.J. Golin, G.F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the Internet. In *Proceedings of IEEE INFOCOM'99*, volume 3, pages 1282–1290, New York, 1999.

T.V. Nguyen, F. Safaei, P. Boustead, and C.T. Chou. Provisioning overlay distribution networks. *Comput Netw*, 49:103–118, 2005.

H. Pirkul. An integer programming model for the allocation of databases in a distributed computer system. *Eur J Oper Res*, 26:401–411, 1986.

L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM'01*, volume 3, pages 1587–1596, 2001.

P. Radoslavov, R. Govindan, and D. Estrin. Topology informed Internet replica placement. *Comput Commun*, 25:384–392, 2002.

J. Ryoo and S.S. Panwar. File distribution in networks with multimedia storage servers. *Networks*, 38:140–149, 2001.

S. Saroiu, K.P. Gummadi, R.J. Dunn, S.D. Gribble, and H.M. Levy. An analysis of Internet content delivery systems. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, December 2002.

T. Wauters, J. Coppens, F. De Turck, B. Dhoedt, and P. Demeester. Replica placement in ring based content delivery networks. *Comput Commun*, 2006. In press.

J. Xu, B. Li, and D.L. Lee. Placement problems for transparent data replication proxy services. *IEEE Journal on Selected Areas in Communications*, 20:1383–1398, 2002.

Z. Xuanping, W. Weidong, T. Xiaopeng, and Z. Yonghu. *Data Replication at Web Proxies in Content Distribution Network*, volume 2642 of *Lecture Notes in Computer Science*, pages 560–569. Springer-Verlag, Xian, 2003.

M. Yang and Z. Fei. A model for replica placement in content distribution networks for multimedia applications. In *Proceedings of IEEE International Conference on Communications (ICC '03)*, volume 1, pages 557 –561, 2003.

G.K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.

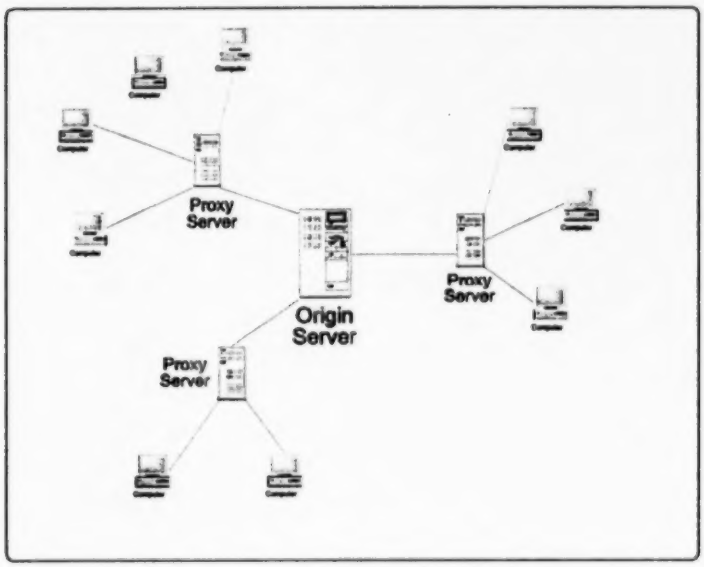

Figure 1: A typical CDN architecture with three proxy servers and nine clients

Table 1: Summary of the notation used for the CDN model

Sets	
I	set of clients
J	set of proxy servers
K	set of objects
S	set containing the origin server
Parameters	
b_k	size of object $k \in K$
s_j	capacity of the proxy server on node $j \in V$
λ_{ik}	probability of client $i \in I$ requesting object $k \in K$
c_{ij}	unit cost of transferring an object over a logical link $\{i,j\} \in E$
d_{ij}	unit delay caused by transferring an object over a logical link $\{i,j\} \in E$
$t(b_k, d_{ij})$	time required to transfer object k from node j to node i
$t'(b_k, d_{ij})$	time required to transfer object k from the origin server to node i via server j
Δ_{ik}	upper bound on the end-to-end delay for client i to access object k

Table 2: Latency parameters for the objects

Group	size (KB)	u_i^k (ms)	Δ_d (ms)
1	$1 \leq b_k \leq 15$	20	100
2	$16 \leq b_k \leq 31$	30	200
3	$32 \leq b_k \leq 63$	40	400
4	$64 \leq b_k \leq 100$	50	600

Table 3: Best results obtained by various policies at the first level

$\|J\|$	$\|I\|$	$\|K\|$	$v(b,\phi)$	$v(b,\rho)$	$v(b,\pi)$	$v(b,\phi)$	$v(\psi,\phi)$	$v(\psi,\rho)$	$v(\psi,\pi)$	$v(\psi,\phi)$	$v(\pi,\phi)$	$v(\pi,\rho)$	$v(\pi,\pi)$	$v(\pi,\phi)$
20	200	60							+					
30	200	60							+					
40	200	60							+					
50	200	60							+					
60	200	60							+					
20	200	80							+					
30	200	80						+						
40	200	80							+					
50	200	80					+							
60	200	80							+					
20	200	100							+					
30	200	100							+					
40	200	100						+						
50	200	100							+					
60	200	100						+						

Table 4: Best results obtained by various policies at the second level

$\|J\|$	$\|I\|$	$\|K\|$	$v(b,\phi)$	$v(b,\rho)$	$v(b,\pi)$	$v(b,\phi)$	$v(\psi,\phi)$	$v(\psi,\rho)$	$v(\psi,\pi)$	$v(\psi,\phi)$	$v(\pi,\phi)$	$v(\pi,\rho)$	$v(\pi,\pi)$	$v(\pi,\phi)$
20	200	60						+						
30	200	60										+		
40	200	60							+					
50	200	60										+		
60	200	60										+		
20	200	80							+					
30	200	80										+		
40	200	80											+	
50	200	80										+		
60	200	80										+		
20	200	100									+			
30	200	100									+			
40	200	100										+		
50	200	100					+							
60	200	100							+					

Table 5: Effect of assigning different probabilities to two neighbourhoods in the one-level SA algorithm on the solution value

$\|J\|$	$\|I\|$	$\|K\|$	.1/.9	.2/.8	.3/.7	.4/.6	.5/.5	.6/.4	.7/.3	.8/.2	.9/.1
20	200	60	28186.34	28274.17	28105.86	28091.75	28271.80	28271.80	28039.40	28579.36	28216.81
30	200	60	26958.76	26873.36	26873.36	26954.34	26723.43	26736.13	26767.85	26849.10	26722.14
40	200	60	27816.61	27622.66	27836.82	27512.80	27458.92	27481.92	27503.80	27434.76	27431.33
50	200	60	26941.95	26872.73	27033.39	26980.95	26928.59	26795.69	26791.43	26865.77	26664.80
60	200	60	26360.91	26251.72	26253.18	26216.37	26392.68	26368.71	26368.71	26368.71	26173.66
20	200	80	32821.42	32211.05	32205.46	32361.35	32342.10	32247.87	32296.31	32209.21	32205.19
30	200	80	28698.69	28301.33	28020.74	28206.70	28013.94	28007.94	28045.98	28190.14	28064.59
40	200	80	30262.06	30321.77	30321.77	29687.92	29700.86	29661.63	29664.37	29706.93	29685.19
50	200	80	30576.65	30339.35	30497.91	30271.92	30324.15	30591.67	30328.43	30559.77	30559.77
60	200	80	29995.51	29386.81	29545.61	29371.21	29430.09	29328.63	29391.83	29425.78	29425.78
20	200	100	34407.56	34096.57	33819.16	33796.28	33773.68	33804.38	33750.62	33750.80	33750.80
30	200	100	32707.17	31889.04	31638.92	31530.63	31641.57	31680.72	31560.85	31675.79	31578.15
40	200	100	30453.11	30474.62	30466.27	30409.88	30428.50	30424.45	30432.31	31929.06	31929.06
50	200	100	32986.24	32682.75	32761.97	32464.45	32407.31	32438.49	32415.94	32426.33	32423.37
60	200	100	33032.23	33264.08	32716.53	32719.26	32699.92	32714.49	32795.04	32737.79	32740.76

Table 6: Comparison of the proposed algorithm with a standard simulated annealing algorithm and "truncated" exact solution methods

$\|J\|$	$\|I\|$	$\|K\|$	v_1	v_2	v_e	$imp(\%)$
20	200	60	28039.40	28010.93	**27207.29**	-2.95
30	200	60	26722.14	**26691.37**	26988.93	1.10
40	200	60	27431.33	**27419.94**	28918.91	5.18
50	200	60	26664.80	**26588.05**	28618.52	7.09
60	200	60	26173.66	**26052.30**	27768.56	6.18
20	200	80	32205.19	32179.67	**31413.67**	-2.44
30	200	80	28007.94	**27906.90**	27963.23	0.20
40	200	80	29661.63	**29556.16**	32247.84	8.35
50	200	80	30271.92	**30188.04**	32506.89	7.13
60	200	80	29328.63	**29323.40**	31664.30	7.39
20	200	100	33750.62	33716.59	**32618.58**	-3.37
30	200	100	31530.63	**31450.86**	32620.78	3.59
40	200	100	30409.88	**30322.23**	32646.99	7.12
50	200	100	32407.31	**32376.74**	33303.75	2.78
60	200	100	32795.04	**32702.77**	33056.24	1.07

Appendix

Table 7: Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(b,\phi)$	$t(b,\phi)$	$v(b,\rho)$	$t(b,\rho)$	$v(b,\pi)$	$t(b,\pi)$	$v(b,\phi)$	$t(b,\phi)$
20	200	60	29747.84	10.06	29747.84	14.37	29747.84	13.85	29747.84	14.05
30	200	60	28929.55	23.21	28929.55	27.15	28857.06	27.41	28857.06	27.64
40	200	60	29808.70	32.59	29812.21	47.43	29804.95	52.58	29804.95	53.22
50	200	60	28578.89	47.04	28578.89	69.97	28578.89	71.05	28578.89	72.51
60	200	60	28284.40	88.38	28284.40	118.23	28284.40	116.45	28284.40	119.06
20	200	80	34880.21	13.75	34842.06	19.32	34880.21	18.95	34880.21	19.61
30	200	80	30486.35	26.78	30486.35	39.68	30468.90	38.73	30468.90	40.33
40	200	80	32239.05	41.63	32239.05	69.75	32199.60	64.66	32199.60	68.22
50	200	80	32838.57	73.25	32791.04	106.04	32764.06	103.28	32764.06	112.67
60	200	80	31936.65	90.00	31942.31	125.39	31942.31	143.95	31942.31	143.97
20	200	100	36901.17	16.30	36901.17	23.26	36901.17	21.38	36901.17	21.88
30	200	100	34535.60	45.12	34535.60	46.87	34526.73	49.72	34526.73	49.89
40	200	100	32965.57	76.96	32965.57	82.36	32965.57	86.25	32965.57	85.02
50	200	100	35345.17	119.81	35330.76	116.02	35308.46	124.80	35308.46	121.19
60	200	100	36120.03	154.89	36120.03	195.07	36120.03	178.53	36120.03	176.59

Table 8: Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(\psi,\phi)$	$t(\psi,\phi)$	$v(\psi,\rho)$	$t(\psi,\rho)$	$v(\psi,\pi)$	$t(\psi,\pi)$	$v(\psi,\phi)$	$t(\psi,\phi)$
20	200	60	28673.58	12.24	28383.54	12.40	28186.91	12.23	28909.07	12.35
30	200	60	27626.13	23.49	26847.65	23.06	26756.60	23.06	27778.54	23.48
40	200	60	28198.51	40.86	27549.02	41.19	27444.75	40.76	28314.25	41.06
50	200	60	27304.31	58.08	26660.49	56.74	26660.90	57.26	27378.61	57.91
60	200	60	27007.31	85.72	26245.13	86.90	26179.41	88.56	27001.79	87.29
20	200	80	32958.09	16.25	32306.51	16.42	32251.83	16.61	33292.08	16.55
30	200	80	28832.71	31.29	28014.50	30.92	28033.37	31.42	28933.03	31.53
40	200	80	30507.29	54.58	29983.11	54.28	29862.18	54.90	30837.48	55.22
50	200	80	31119.67	77.56	31150.92	85.79	31132.19	77.12	31863.40	78.56
60	200	80	30410.48	114.06	29503.12	114.83	29428.42	115.17	30494.42	115.41
20	200	100	34774.70	20.70	33863.23	20.66	33733.65	20.79	35066.60	20.67
30	200	100	32409.63	38.88	31640.14	39.05	31573.66	38.87	32606.52	39.00
40	200	100	31141.83	68.99	30369.89	68.71	30436.65	68.02	31625.85	69.63
50	200	100	33156.16	96.70	32465.88	96.54	32375.50	96.50	33551.70	97.17
60	200	100	33647.26	142.38	32752.54	142.45	34192.94	143.97	33953.02	142.24

Table 9: Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(\pi,\phi)$	$t(\pi,\phi)$	$v(\pi,\rho)$	$t(\pi,\rho)$	$v(\pi,\pi)$	$t(\pi,\pi)$	$v(\pi,\phi)$	$t(\pi,\phi)$
20	200	60	29611.01	12.46	29326.87	12.24	29351.44	12.21	29650.32	8.85
30	200	60	28612.10	23.86	28069.00	22.91	28018.83	22.91	28606.78	16.57
40	200	60	29064.20	41.72	28834.20	39.95	28618.78	40.26	29340.03	28.85
50	200	60	28287.84	58.74	27576.11	48.49	27531.27	56.86	28240.31	40.96
60	200	60	27879.63	84.83	27456.53	61.22	27353.02	85.67	27921.56	60.57
20	200	80	34316.38	16.06	33860.55	11.46	33767.93	16.10	34543.40	11.41
30	200	80	30003.92	30.48	29480.99	22.05	29403.76	30.49	30087.32	22.05
40	200	80	31521.44	54.69	30891.90	38.75	30992.55	54.16	31741.22	39.06
50	200	80	32473.02	79.39	31332.51	55.41	31487.83	76.85	32645.90	75.31
60	200	80	31459.56	114.68	30391.06	81.62	30321.71	112.98	31529.43	113.12
20	200	100	36314.20	20.32	35713.19	15.89	35581.13	20.39	36335.45	20.34
30	200	100	33690.77	38.31	32843.99	38.43	32871.04	38.37	33987.45	38.85
40	200	100	32324.83	67.71	31676.46	67.74	31448.06	48.68	32546.17	67.78
50	200	100	34355.14	96.00	33629.13	96.93	33553.73	69.02	34612.31	95.72
60	200	100	35683.07	140.24	33795.59	141.53	33813.41	101.00	35321.56	141.22

Table 10: Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(b,\phi)$	$t(b,\phi)$	$v(b,\rho)$	$t(b,\rho)$	$v(b,\pi)$	$t(b,\pi)$	$v(b,\phi)$	$t(b,\phi)$
20	200	60	28123.44	85.52	28123.44	78.61	28123.44	89.40	28123.44	79.28
30	200	60	27363.51	128.17	27363.51	123.00	27363.51	131.78	27363.51	149.69
40	200	60	27894.17	170.35	27894.17	175.61	27894.17	178.09	27894.17	213.55
50	200	60	26699.01	189.84	26699.01	218.76	26699.01	236.94	26699.01	258.30
60	200	60	26506.46	240.81	26506.46	241.61	26506.46	314.69	26506.46	298.57
20	200	80	33116.68	93.25	33116.68	94.85	33116.68	111.72	33116.68	115.56
30	200	80	28040.09	144.58	28040.09	143.65	28040.09	202.44	28040.09	196.14
40	200	80	29651.56	203.79	29651.56	199.92	29651.56	255.44	29651.56	251.72
50	200	80	30841.84	304.54	30841.84	253.25	30841.84	333.00	30841.84	337.20
60	200	80	30003.06	363.18	30003.06	365.14	30003.06	463.14	30003.06	356.18
20	200	100	34351.36	141.88	34351.36	138.28	34351.36	196.42	34351.36	124.43
30	200	100	32585.91	211.50	32585.91	212.85	32585.91	310.66	32585.91	211.03
40	200	100	30511.72	264.82	30511.72	285.19	30511.72	328.44	30511.72	307.62
50	200	100	34654.71	325.08	34654.71	346.20	34654.71	390.41	34654.71	461.52
60	200	100	33278.03	438.21	33278.03	436.32	33278.03	478.42	33278.03	488.11

Table 11: Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(\psi,\phi)$	$t(\psi,\phi)$	$v(\psi,\rho)$	$t(\psi,\rho)$	$v(\psi,\pi)$	$t(\psi,\pi)$	$v(\psi,\phi)$	$t(\psi,\phi)$
20	200	60	28054.34	95.11	28010.93	71.76	28034.74	71.94	28168.91	77.79
30	200	60	26722.66	183.97	26745.16	128.07	26738.56	108.82	26757.27	130.11
40	200	60	27489.25	161.63	27481.59	195.60	27419.94	150.64	27776.45	187.54
50	200	60	26710.41	191.71	26746.06	224.69	26766.55	199.42	26759.15	265.89
60	200	60	26128.58	308.82	26100.19	354.46	26110.72	354.89	26111.55	389.95
20	200	80	32223.24	162.51	32191.24	116.25	32179.67	160.11	32264.07	124.54
30	200	80	28228.70	167.11	27947.55	206.57	27961.46	238.38	28166.28	158.33
40	200	80	29672.78	212.33	29582.57	262.68	29625.49	214.64	29654.21	282.30
50	200	80	30262.37	363.07	30272.07	323.87	30237.16	293.77	30343.20	312.06
60	200	80	29459.59	342.66	29481.16	467.83	29384.84	470.40	29456.88	405.39
20	200	100	33817.63	128.81	33728.73	196.02	33771.80	209.59	33801.37	145.34
30	200	100	31547.70	218.53	31487.81	185.57	31467.38	307.57	31489.76	294.99
40	200	100	30383.29	318.25	30359.21	260.93	30416.38	310.42	30440.20	318.40
50	200	100	32378.74	446.18	32392.23	332.87	32415.91	440.12	32406.55	425.03
60	200	100	32728.34	586.72	32719.24	429.01	32702.77	561.03	32759.35	488.53

Table 12: Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm

$\|J\|$	$\|I\|$	$\|K\|$	$v(b,\phi)$	$t(b,\phi)$	$v(b,\rho)$	$t(b,\rho)$	$v(b,\pi)$	$t(b,\pi)$	$v(b,\phi)$	$t(b,\phi)$
20	200	60	28188.19	79.80	28123.62	101.60	28032.90	130.99	28090.07	79.17
30	200	60	27573.14	123.76	26691.37	133.94	26752.27	181.34	26772.22	119.04
40	200	60	27456.75	231.46	28058.56	252.89	27485.57	147.20	27466.56	162.59
50	200	60	26757.04	216.44	26588.05	317.05	26919.74	184.71	26700.77	184.39
60	200	60	26304.73	251.13	26052.30	285.82	26084.96	229.40	26136.22	229.88
20	200	80	32208.86	98.22	32391.44	116.20	32349.12	104.92	32374.62	92.67
30	200	80	28296.39	171.18	27906.90	176.73	28315.07	158.88	28896.67	143.42
40	200	80	29714.57	245.18	29881.92	250.13	29556.16	218.58	29883.26	195.45
50	200	80	30301.16	296.29	30188.04	270.36	30540.01	280.95	30248.59	277.76
60	200	80	29373.61	449.47	29323.40	309.60	29487.49	342.46	29384.42	358.41
20	200	100	33716.59	147.60	33760.33	119.12	33899.65	121.63	33789.33	137.76
30	200	100	31450.86	294.08	31585.60	180.60	31489.13	180.46	31582.61	203.73
40	200	100	30799.51	364.93	30322.23	252.93	30338.80	249.88	30383.17	249.07
50	200	100	32429.53	515.02	32530.65	355.84	32799.59	327.91	32396.19	316.89
60	200	100	32703.67	511.77	32707.98	490.99	32761.41	447.71	32817.45	410.01

Figure Captions and Table Headings

Figure 1. A typical CDN architecture with three proxy servers and nine clients

Table 1. Summary of the notation used for the CDN model

Table 2. Latency parameters for the objects.

Table 3. Best results obtained by various policies at the first level.

Table 4. Best results obtained by various policies at the second level.

Table 6. Comparison of the proposed algorithm with a standard simulated annealing algorithm and "truncated" exact solution methods.

Table 5. Effect of assigning different probabilities to two neighbourhoods in the one-level SA algorithm on the solution value.

Table 7. Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm.

Table 8. Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm.

Table 9. Solution values (v) and the required solution time (t) (in seconds) for different policies at the first level of the simulated annealing algorithm.

Table 10. Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm.

Table 11. Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm.

Table 12. Solution values (v) and the required solution time (t) (in seconds) for different policies at the second level of the simulated annealing algorithm.